Computação Paralela e Distribuída
Trabalho Prático nº2

Group: t04g07

Alberto José Ribeiro da Cunha : up201906325
Gustavo Speranzini Tosi Tavares : up201700129

FEUP : LEIC
03 de Junho de 2022

Introdução **2**

Implementação **2**

Membership Service 2

Storage Service 4

Replication 6

Conclusão **6**

Introdução

O segundo trabalho de CPD consiste no desenvolvimento de uma key-value store. Ou seja, um sistema de armazenamento para objetos diferentes arbitrários, nos quais consistem em um valor, que corresponde ao conteúdo do objeto, e uma key, que é o meio para qual tal objeto será acessado, funcionando como uma hash table, além disto os itens da key-value store são distribuídos em diferentes conjunto de nós. A linguagem em que o projeto foi desenvolvido é Java.

O serviço é prestado pela classe Store que deve ser chamada da seguinte maneira:

```
$ java Store <node_ip>  <Store_port> <IP_mcast_addr> <IP_mcast_port>
```

O node_ip e o Store_port são para serem usados nas comunicações TCP e o IP_mcast_addr e o IP_mcast_port são para serem usados nas comunicações UDP. Apenas o node_ip é que deve ser diferente entre nós distintos.

O serviço é acessado pela classe TestClient que deve ser chamada da seguinte maneira:

```
$ java TestClient <node_ip> <port> <operation> [<opnd>]
```

O node_ip e o port é para serem usados na comunicação TCP, o operation é a operação das 5 dadas que pretende executar e os opnd os argumentos que são precisos para essas funções.

Implementação

1. Membership Service

O serviço disponibiliza 2 funções para a parte da membership de nós:

a. join(): Para adicionar um nó para um conjunto de nós.

   Essa operação consiste em adicionar um nó, originalmente gerado pela Store, a um segundo nó para criar um grupo, ou a um grupo já existente, e atualizar um ficheiro cluster.txt com a informação sobre quais nós pertencem a esse grupo.

b. leave(): Para remover um nó de um conjunto de nós.

Essa operação consiste em remover um nó de um grupo de nós e consequentemente atualizar o ficheiro cluster.txt.

Para que estas funções funcionem é preciso 2 servidores, um TCP e um UDP. Estes 2 servidores vão estar a ouvir em 2 threads executadas em simultâneo, originadas pelo Store. O Membership Protocol vai então ser necessário para assegurar as comunicações entre os diferentes nós e o cliente.

Para o membership protocol funcionar é preciso guardar informação em ficheiros. Estes ficheiros são criados quando for criado um nó novo pela primeira vez. Estes ficheiros são:

- cluster: guarda os ips e as hash-keys dos nós do cluster, guarda sempre na 1ª linha o ip e a key do próprio nó
- log: guarda a informação sobre os últimos eventos de join e leave no cluster
- counter: guarda um counter que indica se o nó faz parte ou não de algum cluster no momento, par faz parte e ímpar não

As comunicações serão feitas através de mensagens de texto, legíveis por um ser humano comum. Estas têm 2 formatos:

1. <char que indica a operação> <argumentos que têm de passar>
2. <resposta para ser lida pelo cliente>

As primeiras são feitas entre para os servidores, o primeiro caráter determina a operação a realizar, sendo o resto argumentos necessários a essa função, que serão depois tratados. As segundas são as respostas enviadas pelos servidores que serão lidas pelo usuário do serviço.

Quando o cliente manda o comando de join, o servidor TCP manda dar multicast do IP, como da hash-key do mesmo. Todos os nós abertos recebem o multicast, mas apenas as que fazem parte do cluster e que não mandaram ainda a informação é que vão processar esta mensagem(fault tolerance). Ao receber este alerta, os nós adicionam o IP ao seu ficheiro de cluster, como o evento de join ao log. Depois, enviam, por TCP, a informação sobre o cluster ao nó que se vai juntar.

O multicast join é reenviado 3 vezes ou até receber informação de pelo menos 3 nós. O servidor impede de receber após 3 mensagens de transferência de informação(fault tolerance). Se não receber nenhuma mensagem nessas 3 tentativas, considera que não existe nenhum cluster, que o cluster não tem nenhuma nó, e cria esse cluster. Após se juntar ou criar o cluster, muda o valor do counter para par.

Para realizar os reenvios do multicast de IP cria-se uma nova thread. Esta inicia um delay para mandar novamente o multicast. Faz isto 2 vezes . Ao fim da 3ª tentativa, se não houver resposta nenhuma, cria um cluster novo.

Quando o cliente envia o comando leave, o servidor começa, também, um multicast do IP com a hash-key. Tal como no join, apenas nós que estejam no cluster, ou seja, tenham o counter par, é que processam esta mensagem. O servidor UCP também garante que o próprio nó não processa a informação. Nesta operação o IP do nó é removido do cluster, e é retirada a mensagem de join do log, substituída por um evento de leave. Esta substituição também existe no join.

No servidor TCP ainda é apagado o conteúdo do ficheiro do log e tira todos os ips que não o próprio do ficheiro de cluster, como, também, muda o valor do counter para ímpar.

Devido a problemas de tempo, não foi possível adicionar a parte de refrescar a informação do cluster com um multicast do log.

2. Storage Service

Para o serviço de armazenamento, o serviço disponibiliza 3 funções:

a. put(key, value): Para adicionar um par key-value para uma store.

   Essa operação tem como base receber um ficheiro com um conteúdo que representa o valor. Ao enviar para a Store, será gerado um ficheiro com o mesmo conteúdo e o nome é a chave gerada através da encriptação SHA-256 do valor do ficheiro.

b. get(key) : Para retornar o valor associado a uma key.

   Essa operação consiste em receber o valor de um ficheiro que está armazenado na Store através de seu nome que é a chave.

c. delete(key): Para deletar um par key-value.

   Essa operação consiste em deletar um ficheiro que representa um par key-value armazenado em uma Store, mas na realidade fazemos o uso de tombstones markers para isto, no qual em vez de deletarmos o ficheiro, renomeamos ele de forma “deleted:key” e removemos o valor do ficheiro.

A primeira função armazena um par key-value numa nó. Esta função funciona independente do nó de que é chamado. Abre o ficheiro de cluster e calcula as distâncias de todas as keys dos ips com a key que recebe como argumento. Ela vai enviar uma mensagem TCP ao nó com a menor distância, que armazena o par num ficheiro cujo nome é a key.

A segunda função recebe a key de um ficheiro e retorna o conteúdo desse mesmo. Funciona como a última função, procura o nó mais próximo e envia-lhe uma mensagem para retornar a informação pretendida e, por sua vez, retorna esta informação para o cliente.

A terceira e última função recebe a key de um ficheiro que pretende deletar. E envia uma mensagem ao servidor do nó mais perto para este eliminar o ficheiro.

A tolerância a falhas nesta parte impede que as operações sejam realizadas caso haja um erro na parte dos ficheiros, seja ler, escrever, criar, etc.

Este serviço é afetado por mudanças no cluster. Ao receber a mensagem de multicast de join, a nó compara a sua distância as keys que fazem parte do si com a distância a este novo nó. Caso a distância seja menor, o par key-value é transferido para o novo nó e eliminado do atual.

No caso de um servidor TCP receber uma mensagem de leave, o nó procura no cluster o próximo nó mais próximo e transfere para lá o par key-value. Faz isto para todos os ficheiros que estejam dentro do nó.

O formato de mensagens usadas nesta parte é idêntico ao formato usado no Membership Service.

3. Replication

A replicação consiste no armazenamento de pares key-value em 3 nós diferentes, para no caso do nó original cair, continuamos com acesso aos pares key-value pertencentes àquele nó em outros nós. As mudanças que isto causa são:

1. put: Agora em vez de guardar apenas uma cópia, guarda-se 3, para isso calcula-se, agora, os 3 nós mais próximos da key do par.
2. get: Não altera muito, apenas se falhar ao ver o primeiro, vai para o segundo e assim para o terceiro (fault tolerance).

3. delete: Em vez de eliminar o ficheiro, agora marca o par como efetivamente eliminado, uma tombstone. Simplesmente adiciona “deleted:” a frente do nome do ficheiro.
4. join: Em vez de fazer as comparações sempre que encontrar um par, faz apenas estas comparações se estiver no terceiro nó mais distante para esse específico par key-value. Não avalia tombstones.
5. leave: Ao procurar uma nó para mandar o par key-value, vai procurar pela nova terceira mais distante. Tal como no join, não vai mandar tombstones.

Conclusão

O grupo foi capaz de atingir o objetivo do trabalho e consolidar o conhecimento dessa cadeira , mas sabe que tem espaço para melhorar e gostaria de ter feito a implementação do protocolo RMI e thread pools.

Participação: Alberto 70%

Gustavo 30%